Implicitly-Defined Neural Networks for Sequence Labeling

Michaeel Kazi, Brian Thompson
MIT Lincoln Laboratory
244 Wood St, Lexington, MA, 02420, USA
{first.last}@ll.mit.edu

Abstract

[1] In this paper, we propose an implicit neural network architecture and show that it can be computed in a reasonably efficient manner. Our architecture relaxes the causality assumption in formulating recurrent neural networks, so that the hidden states of the network are coupled together, in order to improve performance on complex, long-range dependencies in either direction of a sequence. We contrast our architecture with a bidirectional RNN, and show that our proposed architecture the bidirectional network matches it's performance on one task, while providing an ensembling benefit greater than ensembling multiple bidirectional networks.

1 Introduction

Feedforward neural networks were designed to approximate and interpolate functions. Recurrent networks were developed to predict sequences. These recurrent networks can be 'unwrapped', and thought of as a very deep feedforward network, with each layer sharing the same set of weights. Computation proceeds one step at a time, like the trajectory of an ordinary differential equation when solving an initial value problem. However, in certain applications in natural language processing, especially those with long-distance effects, and where grammar matters, sequence prediction may be better thought of as a boundary value problem, because information flows inward from the boundary and creates a strongly coupled system. The bidirectional network addresses this problem by allowing information to flow in both directions. However, this still equates to running two 'for' loops through the data. Many algorithms in practice require more than two passes through the data to determine the answer. We attempt to provide a slightly different mechanism to the bidirectional network where our motivation is a program which iterates over itself until convergence.

[1]Distribution A: Public Release, unlimited distrituon. This work is sponsored by the Air Force Research Laboratory under Air Force contract FA-8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the authors and are not necessarily endorsed by the United States Government.

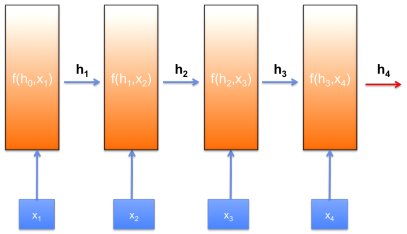

Figure 1: Traditional recurrent neural network structure.

1.1 Related Work

Long-range dependencies have been an issue as long as there have been NLP tasks, and there are many effective approaches to dealing with them. In the context of Hidden Markov models (HMMs), there are the "Forward-Backward" models. In information extraction, there are non-local sequence models that use Gibbs sampling (Finkel et al., 2005), which is the paper most similar to this work. In recent years, the popularity has soared for the Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) and variants such as Gated Recurrent Unit (GRU) (Cho et al., 2014), which enabled recurrent neural networks to process long sequences without the problem of vanishing or exploding gradients, and thus retain information about dependencies. The Bidirectional LSTM (b-LSTM) (Graves and Schmidhuber, 2005), a natural extension of (Schuster and Paliwal, 1997), incorporates past and future hidden states via two separate recurrent networks, allowing information/gradients to flow in two directions.

2 The Implicit Recurrent Neural Network

2.1 Assumptions of Recurrent Neural Networks

A typical recurrent neural network has an input sequence $[x_1, x_2, \ldots, x_s]$ and initial state h_0 and iterativly produces future states:

$$\begin{aligned} h_1 &= f(x_1, h_0) \\ h_2 &= f(x_2, h_1) \\ &\ldots \\ h_s &= f(x_s, h_{s-1}) \end{aligned}$$

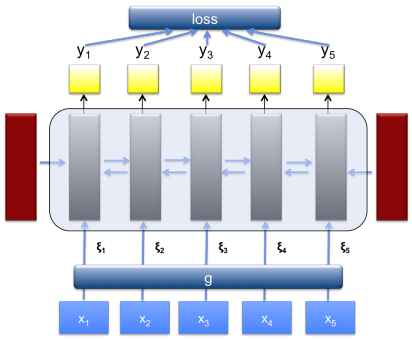

Figure 2: Proposed implicit network architecture

The LSTM, GRU, and related variants follow this formula, with different choices for the state transition function. Computation proceeds left-to-right, with each next state depending only on inputs and previously computed hidden states. In this work, we investigate relaxing this assumption by allowing $h_t = f(x_t, h_{t-1}, h_{t+1})$[2]. This leads to an implicit set of equations for the entire sequence of hidden states, which can be thought of as a single tensor H:

$$H = [h_0, h_1, h_2, \ldots, h_s]$$

This yields a system of nonlinear equations. This setup has the potential for arriving at nonlocal, whole-sequence dependent results. As an additional motivation, we may also wonder if such a system is more 'stable', whereby the predicted sequence may drift less from the true meaning, since errors will not compound with each time step in the same way.

2.2 Proposed Architecture

There are many potential ways to architect a neural network – in fact, this flexibility is one of deep learning's best features – but we restrict our discussion to the one depicted in Figure 2. In this setup, we have the following variables:

data	X
labels	Y
parameters	θ
transformed input	ξ
hidden layers	H

and functions:

loss function	$L = \ell(\theta, H, Y)$
implicit hidden layer definition	$H - F(\theta, \xi, H) = 0$
input layer transformation	$\xi = g(\theta, X)$

Our implicit definition function, F, is made up of local state transitions, and forms a system of nonlinear equations that we need to solve:

[2] A wider stencil can also be used, e.g. $f(h_{t-2}, h_{t-1}, \ldots)$.

$$\begin{aligned} h_1 &= f(h_0, h_2, \xi_1) \\ &\ldots \\ h_i &= f(h_{i-1}, h_{i+1}, \xi_i) \\ &\ldots \\ h_n &= f(h_{n-1}, h_{n+1}, \xi_n) \end{aligned}$$

2.3 Computing the forward pass

To evaluate the network, we must solve the equation $H = F(H)$. We computed this via an approximate Newton solve:

$$H_{n+1} = H_n - (I - \nabla_H F)^{-1}(H_n - F(H_n))$$

Since $(I - \nabla_H F)$ is sparse, we apply Krylov subspace methods (Knoll and Keyes, 2004), specifically BiCG-Stab method (Van der Vorst, 1992), since the system is non-symmetric. This has the added advantage of only relying on matrix-vector multiplies of the gradient of F, which can be conveniently and efficiently computed via the Theano (Bergstra et al., 2011) operator `Rop`.

We also considered approximating the inverse via a polynomial $P_n(\nabla_H F)$, and used the geometric series $P_n(x) = 1 + x + x^2 + \ldots + x^n$, which converges provided that $||\nabla_H F|| < 1$. This, with $n = 1$, proved a reasonable approximation for the task of Part-of-Speech tagging (Section 3.1).

However, for other tasks, we found the eigenvalues of the system were not as well-behaved.

2.4 Gradients

In order to train the model, we perform gradient descent. Taking the gradient of the loss function:

$$\nabla_\theta L = \nabla_\theta \ell + \nabla_H \ell \nabla_\theta H$$

so we will need to know the gradient of the hidden units with respect to the parameters, which we can find via the implicit definition:

$$\begin{aligned} 0 &= \nabla_\theta H - \nabla_\theta F - \nabla_H F \nabla_\theta H - \nabla_\xi F \nabla_\theta \xi \\ (I - \nabla_H F)\nabla_\theta H &= \nabla_\theta F + \nabla_\xi F \nabla_\theta \xi \\ \nabla_\theta H &= (I - \nabla_H F)^{-1} (\nabla_\theta F + \nabla_\xi F \nabla_\theta \xi) \end{aligned}$$

The entire gradient is then

$$\nabla_\theta L = \nabla_\theta \ell + \nabla_H \ell (I - \nabla_H F)^{-1} (\nabla_\theta F + \nabla_\xi F \nabla_\theta \xi)$$

Once again, the inverse of $I - \nabla_H F$ appears, and we can compute it via Krylov subspace methods. Since it is more efficient to compute $\nabla_H \ell (I - \nabla_H F)^{-1}$ first, our solver in this case uses the Theano `Lop` operator.

2.5 Transition Functions

Recall the original GRU equations, with slight notational modifications:

final hidden	$h_t = (1 - z_t)\hat{h}_t + z_t\tilde{h}_t$
candidate hidden	$\tilde{h}_t = \tanh(Wx_t + U(r_t\hat{h}_t) + \tilde{b})$
update weight	$z_t = \sigma(W_z x_t + U_z\hat{h}_t + b_z)$
reset gate	$r_t = \sigma(W_r x_t + U_r\hat{h}_t + b_r)$

To make this implicit and bidirectional, we let $\hat{h}_t$ (defined as h_{t-1} in (Cho et al., 2014)) be a linear combination of previous and future hidden states via the switch variable s. s is in turn determined by a competition between two sigmoidal units s_p and s_n, representing the contributions of the previous and next hidden states, respectively.

state combination	$\hat{h}_t = sh_{t-1} + (1 - s)h_{t+1}$
switch	$s = \frac{s_l}{s_l + s_r}$
previous switch	$s_p = \sigma(W_p x_t + U_p h_{t-1} + b_p)$
next switch	$s_n = \sigma(W_n x_t + U_n h_{t+1} + b_n)$

3 Experiments

3.1 Part-of-speech tagging

Next we apply our model to a real world problem. Part of speech tagging fits naturally in the sequence labeling framework, and has the advantage of a standard dataset that we can use to compare our network with other techniques. To train a part-of-speech tagger, we simply let L be a softmax layer transforming each hidden unit output into a part of speech tag. Initially, ξ consisted only of word vectors for 39,000 case-sensitive vocabulary words. Next, we lowercased the vocabulary words, but added a single feature indicating whether case appeared in the data. Third, we added six additional 'word vector' components to encode the top-2000 most common prefixes and suffixes of words, for affix lengths 2 to 4. Finally, we added in other (binary) features to indicate the presence of numbers, symbols, punctuation, and more rich case data, as used by (Huang et al., 2015).

We trained the Part of Speech (POS) tagger on the Penn Treebank Wall Street Journal corpus (Marcus et al., 1993), blocks 0-18, validated on 19-21, and tested on 22-24, and compared it to the results of the off-the-shelf Stanford Part-of-Speech tagger. The results are indicated in Table 1. We were able to achieve comparable results, and as Manning notes, performance gains past that point are quite difficult, due to errors/inconsistencies in the dataset, ambiguity, and very difficult linguistics, sometimes with dependencies across sentences (Manning, 2011).

Training was done using stochastic gradient descent, with an initial learning rate of 0.5, and a batch size of 20. Word vectors were of dimension 200, prefix and suffix vectors were of dimension 12. Hidden unit size was equal to feature input size, so in this case, 280. Training this way is takes about 5 seconds per batch. Batching the nonlinear solver was slightly tricky – it was straightforward to perform the same BiCG-stab computations across different elements in the batch, but different elements converged quicker than others, and some elements required restarting from a different random initialization. For part-of-speech tagging, however, most of the elements were well-behaved.

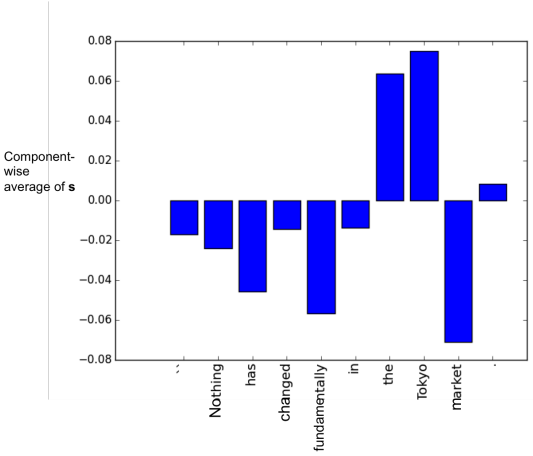

Figure 3: Visualization of the switch variable. Positive values indicate a right-to-left flow of information, while negative values indicate left-to-right. Note that 'Tokyo' is used to modify 'market', instead of being a noun, and thus needs information from 'market' to make the correct determination.

We also visualized some of the outputs of the "switch" variables for various sentences. The switch is made up of many features, so it does not necessarily always correspond to human judgment, but by taking the average, one can get a sense of the flow of information. In Figure 3, we see a visualization of the switch on a very simple sentence, and in Figure 4 we see it in action over a more complicated sentence. Interestingly, phrasal structures emerge.

4 Acknowledgements

This work would not be possible without the support and funding of the Air Force Research Laboratory. We also acknowledge Nick Malyska, Elizabeth Salesky, and Jonathan Taylor at MIT Lincoln Lab for interesting technical discussions related to this work.

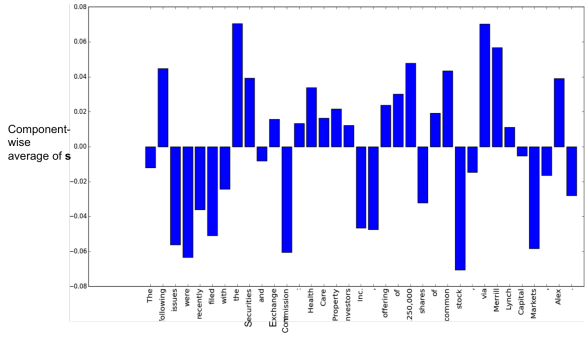

Figure 4: A more complicated example, but that is actually more representative of the types of sentences within the WSJ corpus. Note the entire clause about 'Health Care Property Investors Inc.' propagates information from the right – without the Inc, it may not realize it is the name of a company. Also of note are phrases "offering of 200,000 shares" and "Merrill Lynch Capital Markets." This graphic was done using the case-insensitive model.

Tagger	WSJ Accuracy
Word vectors only	0.9626
Single case feature	0.9650
Ensemble of above (2)	0.9683
Affix word-vectors	0.9714
Case+Symbol feats	0.9730
Ensemble without case	0.9731
Ensemble with case	0.9736
Stanford POS Tagger	0.9732

Table 1: Tagging performance.

Architecture	WSJ Accuracy
GRU	96.51
LSTM	96.53
Bidirectional GRU	97.26
b-LSTM	97.27
Implicit	97.30
b-LSTM Ensemble	97.31
Implicit Ensemble	97.36
Implicit + b-LSTM	97.40

Table 2: Tagging performance relative to other recurrent architectures.

References

James Bergstra, Frédéric Bastien, Olivier Breuleux, Pascal Lamblin, Razvan Pascanu, Olivier Delalleau, Guillaume Desjardins, David Warde-Farley, Ian Goodfellow, Arnaud Bergeron, et al. 2011. Theano: Deep learning on gpus with python. In *NIPS 2011, BigLearning Workshop, Granada, Spain*.

Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder–decoder approaches. *Syntax, Semantics and Structure in Statistical Translation* page 103.

Jenny Rose Finkel, Trond Grenager, and Christopher Manning. 2005. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics, pages 363–370.

Alex Graves and Jürgen Schmidhuber. 2005. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks* 18(5):602–610.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991* .

Dana A Knoll and David E Keyes. 2004. Jacobian-free newton–krylov methods: a survey of approaches and applications. *Journal of Computational Physics* 193(2):357–397.

Christopher D Manning. 2011. Part-of-speech tagging from 97% to 100%: is it time for some linguistics? In *International Conference on Intelligent Text Processing and Computational Linguistics*. Springer, pages 171–189.

Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of english: The penn treebank. *Computational linguistics* 19(2):313–330.

Mike Schuster and Kuldip K Paliwal. 1997. Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on* 45(11):2673–2681.

Henk A Van der Vorst. 1992. Bi-cgstab: A fast and smoothly converging variant of bi-cg for the solution of nonsymmetric linear systems. *SIAM Journal on scientific and Statistical Computing* 13(2):631–644.